Transaction Management and Concurrency Control

Dr. Rabah Mokhtari

Department of Computer Science,
UNIVERSITY OF M'SILA

November 4, 2025

Content

Overview

Overview

Database Transactions

Database transactions reflect real-world transactions that are triggered by events such as buying a product, registering for a course, or making a deposit into a checking account. Transactions are likely to contain many parts, such as updating a customer's account, adjusting product inventory, and updating the seller's accounts receivable.

Transaction Importance

All parts of a transaction must be successfully completed to prevent data integrity problems. Therefore, executing and managing transactions are important database system activities.

What You Will Learn

Transaction Properties

In this presentation you will learn about the main properties of database transactions (atomicity, consistency, isolation, and durability, plus serializability for concurrent transactions).

Topics Covered

- SQL representation of transactions and transaction logs
- Problems with concurrent transactions (lost updates, uncommitted data, inconsistent retrievals)
- Concurrency control algorithms: locks, time stamping, optimistic methods
- Database recovery management for maintaining consistent states

What Is a transaction?

What Is a transaction?

Sales database

To illustrate what transactions are and how they work, use the sales database illustrated in the following relational diagram.

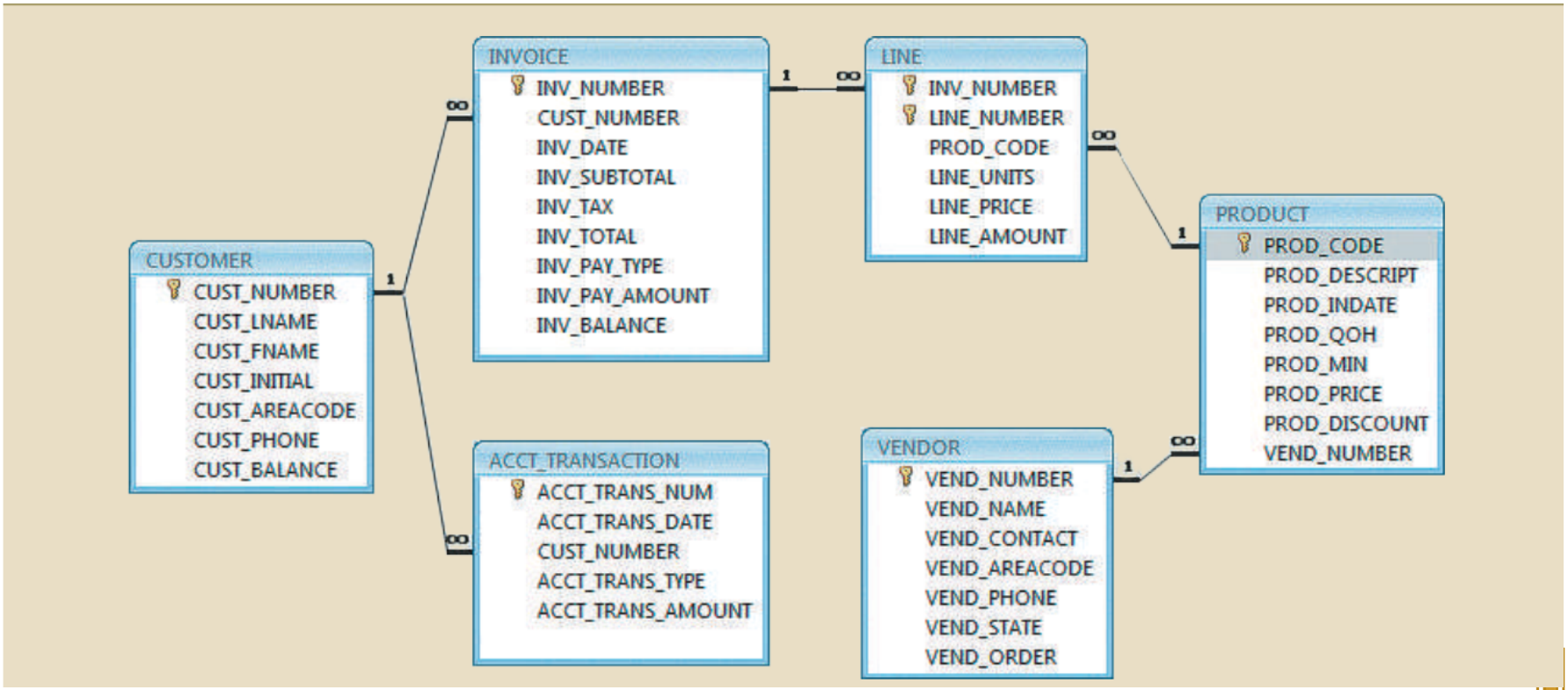

Figure: Sales database

Database Design Features

Examining the Relational Diagram

As you examine the relational diagram of the sales database, note the following features:

1. Customer Balance Storage

- `CUST_BALANCE` is stored in the CUSTOMER table
- Increases with credit purchases, decreases with payments
- Enables easy queries for current balances and summaries
- Facilitates calculation of total, average, min/max balances

2. Transaction Tracking

- ACCT_TRANSACTION table records all purchases and payments
- Tracks detailed customer account activity
- Design could be refined for accounting precision
- Current implementation sufficient for understanding transactions

Understanding Transactions

Sales Transaction Scenario

To understand the concept of a transaction, suppose that you sell a product to a customer who may charge the purchase to their account. This sales transaction consists of:

- Writing a new customer invoice
- Reducing the quantity on hand in inventory
- Updating the account transactions
- Updating the customer balance

Database Transaction

In database terms, a **transaction** is any action that reads from or writes to a database. A transaction may consist of:

- A simple SELECT statement
- A series of related UPDATE statements
- A series of INSERT statements
- A combination of SELECT, UPDATE, and INSERT statements

Transaction Properties

Definition (Transaction)

A **transaction** is a logical unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable. All SQL statements must complete successfully or the entire transaction is rolled back.

Consistent Database State

A successful transaction changes the database from one **consistent database state** to another. A **consistent database state** is one in which all data integrity constraints are satisfied.

- Every transaction must begin with the database in a known consistent state
- Transactions are controlled by the DBMS to guarantee database integrity

Database Requests

Most transactions are formed by two or more **database requests**. A **database request** is equivalent to a single SQL statement.

- Example: 2 UPDATE + 1 INSERT = 3 database requests
- Each request generates several I/O operations

Evaluating Transaction Results

Non-Updating Transactions

Not all transactions update the database. For example, examining a customer balance:

SQL Query

```
SELECT CUST_NUMBER, CUST_BALANCE
FROM CUSTOMER
WHERE CUST_NUMBER = 10016;
```

Transaction Nature

This query represents a transaction because it accesses the database, maintaining consistency without alterations.

Transaction Composition

A transaction may consist of:

- A single SQL statement
- A collection of related SQL statements

Complex Transaction Example

Sales Transaction Scenario

January 18, 2018: Credit sale of two units of product 89-WRE-Q to customer 10016 for $277.55

Initial State (Before Transaction)

CUSTOMER table

CUST_NUMBER	CUST_NAME	CUST_BALANCE
10016	John Smith	500.00
10017	Maria Garcia	320.75

INVOICE table

INV_NUMBER	CUST_NUMBER	INV_DATE	INV_TOTAL
1008	10017	15-Jan-2018	45.00

LINE table

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE
1008	1	45-TYU-L	3	37.50

PRODUCT table

PROD_CODE	PROD_DESCRIPTION	PROD_QOH	PROD_PRICE
89-WRE-Q	Wireless Router	15	256.99
45-TYU-L	Network Cable	42	12.50

ACCT_TRANSACTION table

ACT_NUMBER	ACT_DATE	CUST_NUMBER	ACT_TYPE	ACT_AMOUNT
10006	15-Jan-2018	10017	payment	45.00

Complex Transaction example

SQL Transaction Statements

Required SQL statements to finalize the changes needed for the sale operation are given by the following complex transaction.

```
INSERT INTO INVOICE VALUES (1009, 10016, '18-Jan-2018', 513.98, 0.00, 513.98, 'cred', 0.00, 513.98);
INSERT INTO LINE VALUES (1009, 1, '89-WRE-Q', 2, 256.99, 513.98);
UPDATE PRODUCT SET PROD_QOH = PROD_QOH - 2 WHERE PROD_CODE = '89-WRE-Q';
UPDATE CUSTOMER SET CUST_BALANCE = CUST_BALANCE + 513.98 WHERE CUST_NUMBER = 10016;
INSERT INTO ACCT_TRANSACTION VALUES (10007, '18-Jan-2018', 10016, 'charge', 513.98);
COMMIT;
```

Invoice Amounts Clarification

- INV_SUBTOTAL: Amount before taxes
- INV_TOTAL: Final amount after taxes
- Tax = 0.00 for simplicity

Complex Transaction Example

Sales Transaction Scenario

The results of the successfully completed transaction are shown in the following figure

Final State (After Transaction)

CUSTOMER table

CUST_NUMBER	CUST_NAME	CUST_BALANCE
10016	John Smith	1013.98
10017	Maria Garcia	320.75

INVOICE table

INV_NUMBER	CUST_NUMBER	INV_DATE	INV_TOTAL
1008	10017	15-Jan-2018	45.00
1009	10016	18-Jan-2018	513.98

LINE table

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE
1008	1	45-TYU-L	3	37.50
1009	1	89-WRE-Q	2	513.98

PRODUCT table

PROD_CODE	PROD_DESCRIPTION	PROD QOH	PROD_PRICE
89-WRE-Q	Wireless Router	13	256.99
45-TYU-L	Network Cable	42	12.50

ACCT_TRANSACTION table

ACT_NUMBER	ACT_DATE	CUST_NUMBER	ACT_TYPE	ACT_AMOUNT
10006	15-Jan-2018	10017	payment	45.00
10007	18-Jan-2018	10016	charge	513.98

Transaction Results Analysis

Understanding the Transaction Impact

To better understand the transaction results, note the following changes:

1. New Invoice Record

- A new row 1009 was added to the INVOICE table
- Derived attribute values were stored for: invoice subtotal, tax, invoice total, and invoice balance

2. Line Item Added

- The LINE row for invoice 1009 was added
- Reflects purchase of two units of product 89-WRE-Q at $256.99
- Derived attribute values for line amount were stored

Transaction Results Analysis (Continued)

3. Inventory Update

- Product 89-WRE-Q's quantity on hand (PROD_QOH) was reduced by two
- Changed from 15 to 13 units in the PRODUCT table

4. Customer Balance Adjustment

- Customer balance (CUST_BALANCE) for customer 10016 was updated
- Added $277.55 to the existing balance (from $500.00 to $777.55)

5. Account Transaction Record

- A new row was added to the ACCT_TRANSACTION table
- Reflects the new account transaction number 10007
- Records the charge transaction for customer 10016

COMMIT statement

The COMMIT statement was used to end a successful transaction.

Transaction Failure Scenario

System Failure During Transaction

Suppose the DBMS completes the first three SQL statements but loses power during the fourth statement (UPDATE of CUSTOMER table). The results:

Partial Transaction Execution

- INVOICE and LINE rows were added
- PRODUCT table was updated (inventory reduced)
- Customer 10016 was **not** charged
- No record written in ACCT_TRANSACTION table

Database Recovery

- Database is now in an **inconsistent state**
- Not usable for subsequent transactions
- DBMS will roll back to previous consistent state
- Transaction management ensures database integrity

Semantic Correctness

User Responsibility

- DBMS recovers from system failures automatically
- But transaction must be **semantically correct**
- DBMS cannot guarantee transaction represents real-world event correctly

Example: Incorrect Update

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH + 2 - Should be -2!
WHERE PROD_CODE = '89-WRE-Q';
```

Consequences

- Syntax is correct, but logic is wrong
- Adds 2 units instead of subtracting
- Database becomes inconsistent with real world
- DBMS executes it anyway - user's responsibility

Potential Transaction Errors

Common User Errors

- Updating wrong product (1546-QQ2 instead of 89-WRE-Q)
- Crediting wrong customer (10012 instead of 10016)
- Incorrect arithmetic operations (+ instead of -)
- Missing required operations

Devastating Consequences

- Improper transactions can destroy database integrity
- Financial losses, incorrect inventory, customer disputes
- Loss of business intelligence (BI) and reporting accuracy

DBMS Integrity Enforcement

- Constraints based on business rules
- Automatic referential and entity integrity
- Primary key violation detection
- Error codes for constraint violations

Transaction Properties

ACID Properties

The ACID Test

Each individual transaction must display atomicity, consistency, isolation, and durability. These four properties are sometimes referred to as the ACID test.

1. Atomicity

- Requires that all operations (SQL requests) of a transaction be completed
- If not, the transaction is aborted
- Transaction is treated as a single, indivisible, logical unit of work
- Example: If T1 has four SQL requests, all must complete successfully

2. Consistency

- Indicates the permanence of the database's consistent state
- Transaction takes database from one consistent state to another
- If any part violates integrity constraints, entire transaction is aborted

ACID Properties (Continued)

3. Isolation

- Data used during transaction execution cannot be used by other transactions until completed
- Particularly useful in multiuser database environments
- Example: If T1 is using data item X, no other transaction can access X until T1 ends

4. Durability

- Ensures that once transaction changes are done and committed, they cannot be undone
- Changes persist even in the event of system failure
- Guarantees permanent recording of transaction results

Serializability for Concurrent Transactions

Multiple Transaction Execution

When executing multiple transactions concurrently (e.g., T1, T2, T3), the DBMS must schedule their concurrent execution properly.

Serializability Property

- Each transaction must comply with ACID properties
- Schedule of multiple transactions must exhibit serializability
- Ensures concurrent execution yields consistent results
- Crucial for multiuser and distributed databases

Single vs Multiuser Databases

- Single-user DBMS: Automatically ensures serializability (only one transaction)
- Multiuser DBMS: Must implement controls for serializability and isolation
- Must guard database consistency and integrity

Concurrency Control Importance

Isolation Violation Risks

If several concurrent transactions execute over the same data set and the second transaction updates the database before the first transaction is finished:

- Isolation property is violated
- Database becomes inconsistent
- Data integrity is compromised

DBMS Responsibility

- DBMS must manage transactions using concurrency control techniques
- Must avoid undesirable situations caused by concurrent access
- Ensures proper scheduling and isolation of transactions

Key Requirements

- Atomicity and durability: Required for all DBMS types
- Serializability and isolation: Critical for multiuser DBMS
- All properties work together to maintain database integrity

Transaction Management with SQL

ANSI SQL Transaction Standards

ANSI Standards Overview

The American National Standards Institute (ANSI) has defined standards that govern SQL database transactions. Transaction support is provided by two SQL statements: **COMMIT** and **ROLLBACK**.

Transaction Termination Events

The transaction sequence must continue until one of these events occurs:

- **COMMIT statement**: All changes permanently recorded, transaction ends
- **ROLLBACK statement**: All changes aborted, database rolled back
- **Normal program end**: Equivalent to COMMIT
- **Abnormal termination**: Equivalent to ROLLBACK

COMMIT Example

Sales Transaction Scenario

Customer buys two units of product 1558-QW1 at $43.99 each (total $87.98) and charges to account:

SQL Transaction with COMMIT

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH - 2
WHERE PROD_CODE = '1558-QW1';
UPDATE CUSTOMER
SET CUST_BALANCE = CUST_BALANCE + 87.98
WHERE CUST_NUMBER = '10011';
COMMIT;
```

Programming Practice

- COMMIT not strictly necessary if UPDATE is last action
- Good practice to include COMMIT at transaction end
- Ensures explicit control over transaction completion

Transaction Initiation Methods

Implicit Transaction Start

- Transaction begins implicitly when first SQL statement is encountered
- Follows ANSI standard approach

Explicit Transaction Declaration

Some SQL implementations use explicit transaction start statements:

- **SQL Server**: `BEGIN TRANSACTION;`
- **Oracle**: `SET TRANSACTION` with properties as parameters
- Allows transaction characteristics to be specified

Implementation Variations

- Not all SQL implementations follow ANSI standard exactly
- Different syntax for transaction management
- Same underlying principles apply

Transaction Log

Transaction Log Overview

Purpose of Transaction Log

A DBMS uses a transaction log to keep track of all transactions that update the database. The DBMS uses this log for recovery requirements triggered by:

- ROLLBACK statements
- Program's abnormal termination
- System failures (network issues, disk crashes)

Automatic Recovery

- Some RDBMSs use logs to recover database to consistent state
- After server failure, Oracle automatically:
  - Rolls back uncommitted transactions
  - Rolls forward committed transactions not yet written to disk
- Required for transactional correctness

Transaction Log Contents

What the Log Stores

While executing transactions, DBMS automatically updates the transaction log with:

- Record for beginning of transaction
- For each transaction component (SQL statement):
  - Type of operation (INSERT, UPDATE, DELETE)
  - Names of affected objects (table names)
  - "Before" and "after" values for updated fields
  - Pointers to previous/next log entries for same transaction
- Ending (COMMIT) of transaction

Processing Overhead

- Using transaction log increases processing overhead
- But ability to restore corrupted database is worth the price

Transaction Log Recovery Process

System Failure Recovery

If a system failure occurs:

- DBMS examines transaction log for uncommitted/incomplete transactions
- Restores (ROLLBACK) database to previous state
- After recovery, writes committed transactions not physically written before failure

ROLLBACK Operation

- If ROLLBACK issued before transaction termination:
- DBMS restores database only for that particular transaction
- Maintains durability of previous transactions
- Committed transactions are never rolled back

Transaction Log Management

Critical Database Component

- Transaction log is a critical part of the database
- Implemented as separate file(s) from actual database files
- Subject to dangers: disk-full conditions, disk crashes

High Availability Measures

- Some implementations support logs on multiple disks
- Reduces consequences of system failure
- Ensures business continuity and data protection

Importance

- Contains some of the most critical data in DBMS
- Essential for disaster recovery and data integrity
- Required for ACID compliance and system reliability

Transaction log example

A TRANSACTION LOG

TRL_ ID	TRX_ NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_ BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				

TRL_ID = Transaction log record ID
TRX_NUM = Transaction number
PTR = Pointer to a transaction log record ID
(*Note*: The transaction number is automatically assigned by the DBMS.)

Concurrency Control

Concurrency Control

Definition

Coordinating the simultaneous execution of transactions in a multiuser database system is known as **concurrency control**.

Objective

The objective of concurrency control is to ensure the **serializability** of transactions in a multiuser database environment.

Importance

- Most concurrency control techniques preserve the **isolation** property
- Simultaneous execution over shared database can create integrity problems
- Essential for maintaining data consistency

Three Main Problems

1. Lost updates
2. Uncommitted data
3. Inconsistent retrievals

Lost Updates Problem

Definition

The **lost update** problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is lost (overwritten by the other transaction).

Illustrative Scenario

- PRODUCT table with PROD_QOH (Quantity on Hand) attribute
- Current PROD_QOH value = 35
- Two concurrent transactions T1 and T2 update PROD_QOH
- One update gets lost due to concurrent execution

TWO CONCURRENT TRANSACTIONS TO UPDATE QOH

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

Serial execution example

Serial execution result

- Serial execution of the transactions under normal circumstances, yielding PROD_QOH = 105.
- The following Figure shows an example of serial execution.

SERIAL EXECUTION OF TWO TRANSACTIONS

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 – 30	
6	T2	Write PROD_QOH	105

Lost Update Problem Analysis

Lost Update Sequence

However, suppose that a transaction can read a product's `PROD_QOH` value from the table before a previous transaction has been committed, using the same product:

- First transaction (T1) has not yet been committed when second transaction (T2) is executed
- T2 still operates on the value 35, and its subtraction yields 5 in memory
- Meanwhile, T1 writes the value 135 to disk
- T1's value is promptly overwritten by T2

LOST UPDATES

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 – 30	
5	T1	Write PROD_QOH (lost update)	135
6	T2	Write PROD_QOH	5

Result

In short, the addition of 100 units is lost during the process.

Uncommitted Data Problem

Definition

The phenomenon of **uncommitted data** occurs when two transactions, T1 and T2, are executed concurrently and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data; thus violating the isolation property of transactions.

Illustrative Scenario

- Using same transactions from lost updates discussion
- T1 has two atomic parts: update inventory (PROD_QOH) and update invoice total (not shown)
- T1 is forced to roll back due to error during invoice total update
- Rollback undoes the inventory update as well

Uncommitted Data Consequences

Problem Demonstration

- T1 transaction is rolled back to eliminate addition of 100 units
- T2 subtracts 30 from the original 35 units
- Correct answer should be 5
- But T2 may have operated on T1's uncommitted data

Isolation Violation

- T2 accesses data that T1 later rolls back
- Violates the isolation property of transactions
- Leads to inconsistent database state
- T2's operations are based on non-persistent changes

TRANSACTIONS CREATING AN UNCOMMITTED DATA PROBLEM

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

Uncommitted Data Consequences (Cont.)

Uncommitted read consequences

Figure shows how the **serial** execution of these transactions yields the correct answer under normal circumstances.

CORRECT EXECUTION OF TWO TRANSACTIONS

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 – 30	
7	T2	Write PROD_QOH	5

Uncommitted Data Consequences (Cont.)

Uncommitted read consequences

Figure shows how the uncommitted data problem can arise when the ROLLBACK is completed afer T2 has begun its execution.

AN UNCOMMITTED DATA PROBLEM

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	PROD_QOH = 135 – 30	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Inconsistent Retrievals Problem

Definition

Inconsistent retrievals occur when a transaction accesses data before and after one or more other transactions finish working with such data.

Scenario

- T1 calculates summary function over a set of data
- T2 updates the same data concurrently
- T1 reads some data before changes, other data after changes
- Yields inconsistent results

Example Conditions

1. T1 calculates total PROD_QOH for all products
2. T2 updates PROD_QOH for two products concurrently

Inconsistent Retrievals Problem (cont.)

RETRIEVAL DURING UPDATE	
TRANSACTION T1	**TRANSACTION T2**
SELECT SUM(PROD_QOH) FROM PRODUCT	UPDATE PRODUCT SET PROD_QOH = PROD_QOH + 10 WHERE PROD_CODE = 1546-QQ2
	UPDATE PRODUCT SET PROD_QOH = PROD_QOH – 10 WHERE PROD_CODE = 1558-QW1
	COMMIT;

Inconsistent Retrievals Illustration

Transaction Details

- T2 corrects typing error: added 10 units to wrong product
- Correction: add 10 to 1546-QQ2, subtract 10 from 1558-QW1
- Initial and final PROD_QOH values should be consistent

The Problem

- T1 reads 1546-QQ2 (25) after T2's WRITE
- T1 reads 1558-QW1 (23) before T2's WRITE
- Computed total: 102 (incorrect) vs. correct total: 92
- Inconsistent retrieval due to interleaved operations

Inconsistent Retrievals Problem (cont.)

TRANSACTION RESULTS: DATA ENTRY CORRECTION

	BEFORE	AFTER
PROD_CODE	PROD_QOH	PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QW1	23	(23 – 10) → 13
2232-QTY	8	8
2232-QWE	6	6
Total	**92**	**92**

Inconsistent Retrievals Problem (cont.)

INCONSISTENT RETRIEVALS

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 – 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

Concurrency Control Necessity

Critical Importance

- Without concurrency control, multiuser databases create havoc
- Inconsistent retrievals lead to wrong business decisions
- Database consistency only guaranteed before/after transactions

Temporary Inconsistency

- Database moves through temporary inconsistent states during transactions
- Especially when updating multiple tables/rows
- Isolation property prevents accessing unreleased data

The Scheduler

Definition

The scheduler is a special DBMS process that establishes the order in which operations are executed within concurrent transactions.

Responsibilities

- Interleaves execution of database operations
- Ensures serializability and isolation of transactions
- Uses concurrency control algorithms (locking, time stamping)
- Determines which transactions are serializable

Importance

- Prevents conflicts when transactions access related/same data
- Establishes correct execution order
- Creates serializable schedule of operations

Scheduler Benefits

Efficiency Improvements

- Prevents first-come, first-served inefficiency
- Reduces CPU wait time for READ/WRITE operations
- Improves overall system response times
- Better utilization of CPU and storage systems

Data Isolation

- Prevents two transactions updating same data simultaneously
- Manages READ/WRITE conflicts effectively
- Ensures transaction isolation property is maintained

Conflict Scenarios

- Conflicts occur when transactions access same data
- At least one operation must be WRITE for conflict
- Scheduler resolves using concurrency control methods

Read/Write Conflict Scenarios

Two operations are in conflict when they access the same data and at least one of them is a WRITE operation.

READ/WRITE CONFLICT SCENARIOS: CONFLICTING DATABASE OPERATIONS MATRIX

	TRANSACTIONS		
	T1	T2	RESULT
Operations	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

Concurrency Control Methods

Three Main Approaches

1. **Locking methods** - Most frequently used
2. **Time stamping methods**
3. **Optimistic methods**

Scheduler's Role

- Implements chosen concurrency control method
- Creates serializable transaction schedules
- Ensures consistent database state
- Maximizes system efficiency and throughput

Key Objective

The scheduler's main job is to create a serializable schedule where interleaved execution yields same results as serial execution.